

OPERAND QUEUE FOR USE IN A FLOATING POINT UNIT
AND METHOD OF OPERATION

TECHNICAL FIELD OF THE INVENTION

5

The present invention is directed, in general, to processing systems and, more specifically, to an operand queue for use in the floating point unit of a microprocessor.

10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
105
110
115
120
125
130
135
140
145
150
155
160
165
170
175
180
185
190
195
200
205
210
215
220
225
230
235
240
245
250
255
260
265
270
275
280
285
290
295
300
305
310
315
320
325
330
335
340
345
350
355
360
365
370
375
380
385
390
395
400
405
410
415
420
425
430
435
440
445
450
455
460
465
470
475
480
485
490
495
500
505
510
515
520
525
530
535
540
545
550
555
560
565
570
575
580
585
590
595
600
605
610
615
620
625
630
635
640
645
650
655
660
665
670
675
680
685
690
695
700
705
710
715
720
725
730
735
740
745
750
755
760
765
770
775
780
785
790
795
800
805
810
815
820
825
830
835
840
845
850
855
860
865
870
875
880
885
890
895
900
905
910
915
920
925
930
935
940
945
950
955
960
965
970
975
980
985
990
995
1000
1005
1010
1015
1020
1025
1030
1035
1040
1045
1050
1055
1060
1065
1070
1075
1080
1085
1090
1095
1100
1105
1110
1115
1120
1125
1130
1135
1140
1145
1150
1155
1160
1165
1170
1175
1180
1185
1190
1195
1200
1205
1210
1215
1220
1225
1230
1235
1240
1245
1250
1255
1260
1265
1270
1275
1280
1285
1290
1295
1300
1305
1310
1315
1320
1325
1330
1335
1340
1345
1350
1355
1360
1365
1370
1375
1380
1385
1390
1395
1400
1405
1410
1415
1420
1425
1430
1435
1440
1445
1450
1455
1460
1465
1470
1475
1480
1485
1490
1495
1500
1505
1510
1515
1520
1525
1530
1535
1540
1545
1550
1555
1560
1565
1570
1575
1580
1585
1590
1595
1600
1605
1610
1615
1620
1625
1630
1635
1640
1645
1650
1655
1660
1665
1670
1675
1680
1685
1690
1695
1700
1705
1710
1715
1720
1725
1730
1735
1740
1745
1750
1755
1760
1765
1770
1775
1780
1785
1790
1795
1800
1805
1810
1815
1820
1825
1830
1835
1840
1845
1850
1855
1860
1865
1870
1875
1880
1885
1890
1895
1900
1905
1910
1915
1920
1925
1930
1935
1940
1945
1950
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020
2025
2030
2035
2040
2045
2050
2055
2060
2065
2070
2075
2080
2085
2090
2095
2100
2105
2110
2115
2120
2125
2130
2135
2140
2145
2150
2155
2160
2165
2170
2175
2180
2185
2190
2195
2200
2205
2210
2215
2220
2225
2230
2235
2240
2245
2250
2255
2260
2265
2270
2275
2280
2285
2290
2295
2300
2305
2310
2315
2320
2325
2330
2335
2340
2345
2350
2355
2360
2365
2370
2375
2380
2385
2390
2395
2400
2405
2410
2415
2420
2425
2430
2435
2440
2445
2450
2455
2460
2465
2470
2475
2480
2485
2490
2495
2500
2505
2510
2515
2520
2525
2530
2535
2540
2545
2550
2555
2560
2565
2570
2575
2580
2585
2590
2595
2600
2605
2610
2615
2620
2625
2630
2635
2640
2645
2650
2655
2660
2665
2670
2675
2680
2685
2690
2695
2700
2705
2710
2715
2720
2725
2730
2735
2740
2745
2750
2755
2760
2765
2770
2775
2780
2785
2790
2795
2800
2805
2810
2815
2820
2825
2830
2835
2840
2845
2850
2855
2860
2865
2870
2875
2880
2885
2890
2895
2900
2905
2910
2915
2920
2925
2930
2935
2940
2945
2950
2955
2960
2965
2970
2975
2980
2985
2990
2995
3000
3005
3010
3015
3020
3025
3030
3035
3040
3045
3050
3055
3060
3065
3070
3075
3080
3085
3090
3095
3100
3105
3110
3115
3120
3125
3130
3135
3140
3145
3150
3155
3160
3165
3170
3175
3180
3185
3190
3195
3200
3205
3210
3215
3220
3225
3230
3235
3240
3245
3250
3255
3260
3265
3270
3275
3280
3285
3290
3295
3300
3305
3310
3315
3320
3325
3330
3335
3340
3345
3350
3355
3360
3365
3370
3375
3380
3385
3390
3395
3400
3405
3410
3415
3420
3425
3430
3435
3440
3445
3450
3455
3460
3465
3470
3475
3480
3485
3490
3495
3500
3505
3510
3515
3520
3525
3530
3535
3540
3545
3550
3555
3560
3565
3570
3575
3580
3585
3590
3595
3600
3605
3610
3615
3620
3625
3630
3635
3640
3645
3650
3655
3660
3665
3670
3675
3680
3685
3690
3695
3700
3705
3710
3715
3720
3725
3730
3735
3740
3745
3750
3755
3760
3765
3770
3775
3780
3785
3790
3795
3800
3805
3810
3815
3820
3825
3830
3835
3840
3845
3850
3855
3860
3865
3870
3875
3880
3885
3890
3895
3900
3905
3910
3915
3920
3925
3930
3935
3940
3945
3950
3955
3960
3965
3970
3975
3980
3985
3990
3995
4000
4005
4010
4015
4020
4025
4030
4035
4040
4045
4050
4055
4060
4065
4070
4075
4080
4085
4090
4095
4100
4105
4110
4115
4120
4125
4130
4135
4140
4145
4150
4155
4160
4165
4170
4175
4180
4185
4190
4195
4200
4205
4210
4215
4220
4225
4230
4235
4240
4245
4250
4255
4260
4265
4270
4275
4280
4285
4290
4295
4300
4305
4310
4315
4320
4325
4330
4335
4340
4345
4350
4355
4360
4365
4370
4375
4380
4385
4390
4395
4400
4405
4410
4415
4420
4425
4430
4435
4440
4445
4450
4455
4460
4465
4470
4475
4480
4485
4490
4495
4500
4505
4510
4515
4520
4525
4530
4535
4540
4545
4550
4555
4560
4565
4570
4575
4580
4585
4590
4595
4600
4605
4610
4615
4620
4625
4630
4635
4640
4645
4650
4655
4660
4665
4670
4675
4680
4685
4690
4695
4700
4705
4710
4715
4720
4725
4730
4735
4740
4745
4750
4755
4760
4765
4770
4775
4780
4785
4790
4795
4800
4805
4810
4815
4820
4825
4830
4835
4840
4845
4850
4855
4860
4865
4870
4875
4880
4885
4890
4895
4900
4905
4910
4915
4920
4925
4930
4935
4940
4945
4950
4955
4960
4965
4970
4975
4980
4985
4990
4995
5000
5005
5010
5015
5020
5025
5030
5035
5040
5045
5050
5055
5060
5065
5070
5075
5080
5085
5090
5095
5100
5105
5110
5115
5120
5125
5130
5135
5140
5145
5150
5155
5160
5165
5170
5175
5180
5185
5190
5195
5200
5205
5210
5215
5220
5225
5230
5235
5240
5245
5250
5255
5260
5265
5270
5275
5280
5285
5290
5295
5300
5305
5310
5315
5320
5325
5330
5335
5340
5345
5350
5355
5360
5365
5370
5375
5380
5385
5390
5395
5400
5405
5410
5415
5420
5425
5430
5435
5440
5445
5450
5455
5460
5465
5470
5475
5480
5485
5490
5495
5500
5505
5510
5515
5520
5525
5530
5535
5540
5545
5550
5555
5560
5565
5570
5575
5580
5585
5590
5595
5600
5605
5610
5615
5620
5625
5630
5635
5640
5645
5650
5655
5660
5665
5670
5675
5680
5685
5690
5695
5700
5705
5710
5715
5720
5725
5730
5735
5740
5745
5750
5755
5760
5765
5770
5775
5780
5785
5790
5795
5800
5805
5810
5815
5820
5825
5830
5835
5840
5845
5850
5855
5860
5865
5870
5875
5880
5885
5890
5895
5900
5905
5910
5915
5920
5925
5930
5935
5940
5945
5950
5955
5960
5965
5970
5975
5980
5985
5990
5995
6000
6005
6010
6015
6020
6025
6030
6035
6040
6045
6050
6055
6060
6065
6070
6075
6080
6085
6090
6095
6100
6105
6110
6115
6120
6125
6130
6135
6140
6145
6150
6155
6160
6165
6170
6175
6180
6185
6190
6195
6200
6205
6210
6215
6220
6225
6230
6235
6240
6245
6250
6255
6260
6265
6270
6275
6280
6285
6290
6295
6300
6305
6310
6315
6320
6325
6330
6335
6340
6345
6350
6355
6360
6365
6370
6375
6380
6385
6390
6395
6400
6405
6410
6415
6420
6425
6430
6435
6440
6445
6450
6455
6460
6465
6470
6475
6480
6485
6490
6495
6500
6505
6510
6515
6520
6525
6530
6535
6540
6545
6550
6555
6560
6565
6570
6575
6580
6585
6590
6595
6600
6605
6610
6615
6620
6625
6630
6635
6640
6645
6650
6655
6660
6665
6670
6675
6680
6685
6690
6695
6700
6705
6710
6715
6720
6725
6730
6735
6740
6745
6750
6755
6760
6765
6770
6775
6780
6785
6790
6795
6800
6805
6810
6815
6820
6825
6830
6835
6840
6845
6850
6855
6860
6865
6870
6875
6880
6885
6890
6895
6900
6905
6910
6915
6920
6925
6930
6935
6940
6945
6950
6955
6960
6965
6970
6975
6980
6985
6990
6995
7000
7005
7010
7015
7020
7025
7030
7035
7040
7045
7050
7055
7060
7065
7070
7075
7080
7085
7090
7095
7100
7105
7110
7115
7120
7125
7130
7135
7140
7145
7150
7155
7160
7165
7170
7175
7180
7185
7190
7195
7200
7205
7210
7215
7220
7225
7230
7235
7240
7245
7250
7255
7260
7265
7270
7275
7280
7285
7290
7295
7300
7305
7310
7315
7320
7325
7330
7335
7340
7345
7350
7355
7360
7365
7370
7375
7380
7385
7390
7395
7400
7405
7410
7415
7420
7425
7430
7435
7440
7445
7450
7455
7460
7465
7470
7475
7480
7485
7490
7495
7500
7505
7510
7515
7520
7525
7530
7535
7540
7545
7550
7555
7560
7565
7570
7575
7580
7585
7590
7595
7600
7605
7610
7615
7620
7625
7630
7635
7640
7645
7650
7655
7660
7665
7670
7675
7680
7685
7690
7695
7700
7705
7710
7715
7720
7725
7730
7735
7740
7745
7750
7755
7760
7765
7770
7775
7780
7785
7790
7795
7800
7805
7810
7815
7820
7825
7830
7835
7840
7845
7850
7855
7860
7865
7870
7875
7880
7885
7890
7895
7900
7905
7910
7915
7920
7925
7930
7935
7940
7945
7950
7955
7960
7965
7970
7975
7980
7985
7990
7995
8000
8005
8010
8015
8020
8025
8030
8035
8040
8045
8050
8055
8060
8065
8070
8075
8080
8085
8090
8095
8100
8105
8110
8115
8120
8125
8130
8135
8140
8145
8150
8155
8160
8165
8170
8175
8180
8185
8190
8195
8200
8205
8210
8215
8220
8225
8230
8235
8240
8245
8250
8255
8260
8265
8270
8275
8280
8285
8290
8295
8300
8305
8310
8315
8320
8325
8330
8335
8340
8345
8350
8355
8360
8365
8370
8375
8380
8385
8390
8395
8400
8405
8410
8415
8420
8425
8430
8435
8440
8445
8450
8455
8460
8465
8470
8475
8480
8485
8490
8495
8500
8505
8510
8515
8520
8525
8530
8535
8540
8545
8550
8555
8560
8565
8570
8575
8580
8585
8590
8595
8600
8605
8610
8615
8620
8625
8630
8635
8640
8645
8650
8655
8660
8665
8670
8675
8680
8685
8690
8695
8700
8705
8710
8715
8720
8725
8730
8735
8740
8745
8750
8755
8760
8765
8770
8775
8780
8785
8790
8795
8800
8805
8810
8815
8820
8825
8830
8835
8840
8845
8850
8855
8860
8865
8870
8875
8880
8885
8890
8895
8900
8905
8910
8915
8920
8925
8930
8935
8940
8945
8950
8955
8960
8965
8970
8975
8980
8985
8990
8995
9000
9005
9010
9015
9020
9025
9030
9035
9040
9045
9050
9055
9060
9065
9070
9075
9080
9085
9090
9095
9100
9105
9110
9115
9120
9125
9130
9135
9140
9145
9150
9155
9160
9165
9170
9175
9180
9185
9190
9195
9200
9205
9210
9215
9220
9225
9230
9235
9240
9245
9250
9255
9260
9265
9270
9275
9280
9285
9290
9295
9300
9305
9310
9315
9320
9325
9330
9335
9340
9345
9350
9355
9360
9365
9370
9375
9380
9385
9390
9395
9400
9405
9410
9415
9420
9425
9430
9435
9

*Sub
AI*

mathematical operations, such as multiplication and division, cause significant delays during program execution. A pipelined floating point unit (FPU) may be particularly susceptible to long delays during the execution of certain sequences of instructions. For example, a floating point "load" instruction may occur in a pipelined FPU immediately after, or shortly after, a floating point store instruction occurs. This is sometimes referred to as a "read-after-write" (RAW) hazard. The write (or store) operation to system memory may have a long latency before the write data is "committed" to system memory by the processor. The read (or load) operation following the write (or store) operation may occur before the write operation is complete and may, therefore, suffer significant delays waiting for the write operation is complete before the committed data may be read back from memory.

Therefore, there is a need in the art for improved microprocessor that executes mathematical operations more rapidly. In particular, there is a need for an improved floating point unit that executes floating point operations as rapidly as possible. More particularly, there is a need in the art for a floating point unit that minimizes delays caused by writing data to memory.

SUMMARY OF THE INVENTION

The limitations inherent in the prior art described above are overcome by an improved floating point unit for use in a data processor. According to an advantageous embodiment of the present invention, the floating point unit comprises: 1) a plurality of floating point processing units capable of executing floating point instructions that write operands to an external memory and capable of executing floating point instructions that read operands from the external memory; and 2) an operand queue capable of storing a plurality of operands associated with one or more operations being processed in the floating point unit, wherein the operand queue stores a first operand being written to an external memory by a floating point write instruction executed by a first one of the plurality of floating point processing units and wherein the operand queue supplies the first operand to a floating point read instruction executed by a second one of the plurality of floating point processing units subsequent to the execution of the floating point write instruction.

In one embodiment of the present invention, the floating point unit further comprises a store conversion unit capable of converting operands in the plurality of floating point processing units from an internal format associated with the plurality of

floating point processing units to an external format associated with the external memory.

In another embodiment of the present invention, the operand queue receives the first operand from the store conversion unit and transfers the first operand to the external memory.

In still another embodiment of the present invention, the floating point unit further comprises a load conversion unit capable of converting incoming operands received from the external memory from an external format associated with the external memory to an internal format associated with the plurality of floating point processing units.

In yet another embodiment of the present invention, the operand queue receives the incoming operands from the external memory and transfers the incoming operands to the load conversion unit.

gus A2 In a further embodiment of the present invention, the data in the external memory is accessed in groups of N bytes and wherein the floating point unit further comprises at least one aligner capable of receiving a first incoming operand that is misaligned with respect to a boundary between a first N byte group and a second N byte group and aligning the first incoming operand.

In a still further embodiment of the present invention, the operand queue receives the aligned first incoming operand from the at least one aligner.

In a yet further embodiment of the present invention, the at 5 least one aligner sets at least one bit in the operand queue to indicate that the aligned first incoming operand is valid.

The foregoing has outlined rather broadly the features and technical advantages of the present invention so that those skilled in the art may better understand the detailed description of the invention that follows. Additional features and advantages of the invention will be described hereinafter that form the subject of the claims of the invention. Those skilled in the art should appreciate that they may readily use the conception and the specific embodiment disclosed as a basis for modifying or designing other structures for carrying out the same purposes of the present invention. Those skilled in the art should also realize that such equivalent constructions do not depart from the spirit and scope of the invention in its broadest form.

Before undertaking the DETAILED DESCRIPTION OF THE INVENTION, 20 it may be advantageous to set forth definitions of certain words and phrases used throughout this patent document: the terms "include" and "comprise," as well as derivatives thereof, mean inclusion without limitation; the term "or," is inclusive, meaning

and/or; the phrases "associated with" and "associated therewith," as well as derivatives thereof, may mean to include, be included within, interconnect with, contain, be contained within, connect to or with, couple to or with, be communicable with, cooperate with, 5 interleave, juxtapose, be proximate to, be bound to or with, have, have a property of, or the like; and the term "controller" means any device, system or part thereof that controls at least one operation, such a device may be implemented in hardware, firmware or software, or some combination of at least two of the same. It 10 should be noted that the functionality associated with any particular controller may be centralized or distributed, whether locally or remotely. Definitions for certain words and phrases are provided throughout this patent document, those of ordinary skill in the art should understand that in many, if not most instances, such definitions apply to prior, as well as future uses of such 15 defined words and phrases.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention, reference is now made to the following descriptions taken in conjunction with the accompanying drawings, in which:

FIGURE 1 is a block diagram of an exemplary processing system, which includes an integrated microprocessor according to one embodiment of the present invention;

FIGURE 2 illustrates selected portions of the exemplary CPU in greater detail according to one embodiment of the present invention;

FIGURE 3 illustrates selected portions of the exemplary floating point unit in greater detail according to one embodiment of the present invention;

FIGURE 4 illustrates the exemplary operand queue and related portions of the exemplary floating point unit in greater detail according to one embodiment of the present invention; and

FIGURE 5 is a flow chart illustrating the operation of a floating point unit containing an operand queue according to one embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

FIGURES 1 through 5, discussed below, and the various embodiments used to describe the principles of the present invention in this patent document are by way of illustration only and should not be construed in any way to limit the scope of the invention. Those skilled in the art will understand that the principles of the present invention may be implemented in any suitably arranged processing system.

FIGURE 1 illustrates processing system 10, which includes integrated microprocessor 100, according to one embodiment of the present invention. Integrated microprocessor 100 comprises central processing unit (CPU) 105, which has dual integer and dual floating point execution units, separate load/store and branch units, and L1 instruction and data caches. Microprocessor 100 also comprises graphics unit 110, system memory controller 115, and L2 cache 120, which is shared by CPU 105 and graphics unit 110. Graphics unit 110, system memory controller 115, and L2 cache 120 may be integrated onto the same die as CPU 105. Bus interface unit 125 couples CPU 105, graphics unit 110, and L2 cache 120 to memory controller 115. Bus interface unit 125 also may be integrated onto the same die as CPU 105.

Su
A3
Integrated memory controller 115 bridges microprocessor 100 to system memory 140, and may provide data compression and/or decompression to reduce bus traffic over external memory bus 145 which preferably, although not exclusively, has a RAMbus™, fast SDRAM or other type protocol. Integrated graphics unit 110 provides TFT, DSTN, RGB, and other types of video output to drive display 150.

Bus interface unit 125 connects microprocessor 100 through I/O interface 130 to PCI bridge 155, which has a conventional peripheral component interconnect (PCI) bus interface on PCI bus 160 to one or more peripherals, such as sound card 162, LAN controller 164, and disk drive 166, among others. Bus interface unit 125 also connects fast serial link 180 and relatively slow I/O port 185 to microprocessor 100 (via I/O interface 130 and PCI bridge 155). Fast serial link 180 may be, for example, an IEEE 1394 bus (i.e., "Firewire") and/or a universal serial bus ("USB"). I/O port 185 is used to connect peripherals to microprocessor 100, such as keyboard 190 and/or a mouse. In some embodiments, PCI bridge 155 may integrate local bus functions such as sound, disk drive control, modem, network adapter, and the like.

FIGURE 2 illustrates selected portions of CPU 105 in greater detail according to one embodiment of the present invention.

CPU 105 comprises instruction cache 205, instruction

decode/microcode (Ucode) logic 210, integer unit 215, data cache 220, and floating point unit (FPU) 230. FPU 230 is connected to the rest of CPU 105 via four sets of instruction buses 235, two load operand buses 240, and a store operand bus 245. Execution pipelines 215 comprises one or more integer execution (EX) units, address calculation (AC) units, and condition code (CC) units, which are used to set flags in CPU 105.

FPU 230 is an execution unit in the same way that the EX and AC units in execution pipelines 215 are execution units. In one embodiment of the present invention, when an instruction is decoded in instruction decode/Ucode logic 210, it may be broken down into up to, for example, five (5) nodes (or operations). Three of the nodes are EX, AC and CC nodes used by integer unit 215. The other two nodes are floating point nodes: an Add/Multiply/Store node and a load node. If the instruction is an integer instruction that does not use FPU 230, instruction decode/Ucode logic 210 may issue "no-ops" to both nodes. If the instruction is a floating point instruction that uses FPU 230, then two nodes (operations) can be issued.

FPU 230 receives data operands on load (read) operations from data cache 220 over two load operand buses 240. Loaded data from data cache 220 is sent to FPU 230 in the same alignment format as the loaded data is stored system memory 140. As will be explained

below in greater detail, if the load data crosses a line boundary, the load data is sent in two pieces and is aligned prior to being placed in FPU 230 in an operand queue in accordance with the principles of the present invention. Data cache 220 does not guarantee the order or sequentiality of data transfers for parts of a load operation. FPU 230 is responsible for assembling and aligning the load data.

~~Suy AY~~
In the exemplary embodiment, FPU 230 uses two load buses because the frequency of load operations is twice the frequency of floating point operations. Therefore, in order to achieve an execution rate of one floating point operation per clock, FPU 230 uses two load buses 240. FPU 230 uses one store bus 245 to store results to system memory 140 at commit time. Unlike load operations, where the memory alignment is done in FPU 230, rotating data to put it in memory format is done in data cache 220. The reason for one store bus is that store operations only comprise between 5% and 15% of all floating point instructions, so one bus is sufficient for bandwidth purposes.

FPU 230 also contains a write buffer (explained below in greater detail) which allows nodes (operations) with multiple stores in them to be committed to system memory 140 in one cycle. The write buffer contains only the data portion of a floating point store operation.

FIGURE 3 illustrates selected portions of floating point unit 230 in greater detail according to one embodiment of the present invention. Floating point unit (FPU) 230 comprises FPU micro-ROM (UROM) 302, node exchange (XCH)/register mapping logic and logical-to-physical register file (LRF) logic 304, adder 311, multiplier 313, load conversion units 315a and 315b, and scheduling content addressable memory (CAM) devices 320. FPU 230 also comprises opcode queues 341, 342, and 343, which are associated with adder 311, multiplier 313, and load converter units 315a and 315b, respectively. FPU 230 further comprises store converter unit 317, opcode queue 344, and operand queue 345 according to the principles of the present invention. Finally, FPU 230 comprises virtual commit buffer 350 and write buffer 355.

Sub A5 FPU 230 receives opcodes (instructions) from instruction decoder\Ucode logic 210. Since the number of bits required to control FPU 230 may be quite large, instruction decoder\Ucode logic 210 does not send FPU 230 a micro-word. Instead, instruction decoder\Ucode logic 210 sends index values to FPU micro-ROM (UROM) 302. The index values are represented by the inputs instruction/microcode (IU) index (0) to instruction\code (IU) index (3). UROM 302 outputs consists of an add/multiply operation and a load store operation that are applied to node exchange (XCH)/register mapping logic and logical-to-physical register file

Sub A5

logic 304. XCH/Reg & Mapping and LRF logic 304 computes the physical source and destination addresses in system memory 140 of an operand for each instruction in system memory 140 using register offset values represented by inputs register offset (0) through register offset (3).

5

Once re-mapped register addresses are formed, the data dependencies between nodes are resolved. This operation involves reading the LRF, which delivers an address for the physical register. The physical location of the data may be one of three places. The data may be in committed register file (CRF) 335, in which case the CRF register number and a bit which indicates resident data in the CRF is returned. The CRF holds the committed state of the architectural registers of CPU 105. The data could also be in physical register file (PRF) 330 (also known as a reorder buffer) if the data has been computed but not committed yet. In this case, the PRF location and PRF present values are returned for the operand.

10

Sub A6

Finally, the data may not have been computed yet. In this final case, the dependant instruction is marked as pending and the PRF location where the data will be deposited is returned. The dependant instruction then monitors the result busses and when the result is produced, PRF 330 is read to obtain the data. Once the operation and physical locations of the operands have been

15

SUB A6

generated, the opcodes are loaded into opcode queues 341-344 associated with each functional unit and into a content addressable memory (CAM) which controls the operand valid bits.

There are four major functional units in FPU 239. Adder 311 and multiplier 313 perform the majority of the arithmetic. These operations are fully pipelined and have a latency of three clock cycles and a throughput of one clock cycle. FPU 230 uses two load conversion units 315a and 315b to convert load data from a format stored in system memory 140 to the internal format of FPU 230. Load conversion units 315a and 315b receive operands only from operand queue 345. When all pieces of load data in operand queue 345 are valid, one of load conversion units 315a and 315b is scheduled to convert the load data. The opcode in opcode queue 343 indicates how wide the load data is and what format conversion the load data requires.

There is one store conversion unit 317 in FPU 230. Store conversion unit gets its operands from physical register file (PRF) 330 or committed register file (CRF) 335 or by bypassing a result bus. PRF holds temporary results or uncommitted instructions. The format of store data is converted by store conversion unit 317 from the internal format of FPU 230 to the format of system memory 140 and the converted store data is stored in operand queue 345. Operand queue 345 contains an entry for

every operation in FPU 230. When a store (write) instruction is to be committed to system memory 140, the store data is read from operand queue 345 and is written to virtual commit buffer 350. The read operations from operand queue 345 are sequential and as the store data is transferred to virtual commit buffer 350, the store data is also written into any dependent load instructions (i.e. read-after-write hazards) in operand queue 345. Once an instruction may be committed, FPU 230 transfers the store data from virtual commit buffer 350 to write buffer 355.

FIGURE 4 illustrates exemplary operand queue 345 and related portions of exemplary floating point unit 230 in greater detail according to one embodiment of the present invention. Operand queue 345 receives load data from data cache 220 via input aligner 401a and input aligner 401b. If an FPU instruction loads (reads) data from memory that is not aligned on system memory boundaries, the data is read in two parts and is aligned by input aligner 401a and input aligner 401b.

For example, if system memory 140 is aligned on eight byte boundaries, then the instruction "Load address 2" would read eight (8) bytes of data from address 2 through address 9. To perform this load operation, the eight bytes from address 0 through address 7 (low portion) would be loaded into, for example, input aligner 401a from most significant byte to least significant byte

as follows: 7,6,5,4,3,2,1,0. Input aligner 401a would then rotate the load data by two bytes to the following order: 1,0,7,6,5,4,3,2.

Next, bytes 2 through 7 would be transferred to operand queue 345 and the Valid Low bit would be set to Logic 1. The eight bytes

5 from address 8 through address 15 (high portion) would be loaded into input aligner 401b from most significant byte to least significant byte as follows: 15,14,13,12,11,10,9,8. Input aligner 401a would then rotate the load data by two bytes to the

following order: 9,8,15,14,13,12,11,10. Next, byte 8 and byte 9

would be transferred to operand queue 345 and the Valid High bit would be set to Logic 1. Operand queue 345 would now contain

bytes 9,8,7,6,5,4,3,2 and the Valid High bit and Valid Low bit would both be valid. Only at this time may operand queue 345 be used as a source of load data. It is noted that the order that the

data arrives from memory can be reversed and the operand queue will still function properly. If the high portion is delivered to the operand queue first, then bytes 9 and 8 are written into the operand queue and the Valid High flag is set. The operand will not be considered to be valid yet because the low portion has not been received from memory. Thus, operand queue 345 will wait for the remaining data. When the low portion arrives from memory, bytes

7,6,5,3,2 will be written into operand queue 345 and the Valid Low flag will be set. Once both flags (or valid bits) are set, operand

queue 345 will be considered valid and the load operation can proceed.

Virtual commit buffer 350 further comprises forwarding array 351 of content addressable memory (CAM) locations, and virtual commit tag/exception register 352. Forwarding array 351 is indexed by "forward to" addresses and "forward from" addresses and holds instruction numbers. Operand queue 345 holds data associated with particular instruction numbers. When store (write) instructions are executed by FPU 230, the store data may be converted by store conversion unit 317 and placed in operand queue 345. If the slot that the store data is destined to is virtually committed, the store data may bypass operand queue 345 and be written directly into virtual commit buffer 350.

Before a checkpoint can commit, each store instruction on the checkpoint is read sequentially and is placed into virtual commit buffer 350. As the store data are read, they are written back into any dependent load operations in forwarding array 351. Operand queue 345 may have data written into it via store converter decoder 421, load port decoder 422, and load port decoder 423. Data may be read from operand queue 345 by load scheduler 424 and load scheduler 425. When a checkpoint is finally committed, the store data are transferred from four locations at a time in virtual

commit buffer 350 to write buffer 355. Data are transferred from write buffer 355 to data cache 220 under control of data cache 220.

SAC A1

As used herein, "virtual commit" is the process of transferring store data from operand queue 345 into virtual commit buffer 350, as well as storing virtually committed dat into any dependent load slots in operand queue 345. The process of virtual commit is performed on a slot-by-slot basis in operand queue 345 and virtual commit buffer 350. However, a virtual commit cycle is only required if a slot has a floating-point store in it. Checkpoints that do not have any floating-point stores also require 1 cycle to virtually commit.

If there is a floating-point store in a slot of a checkpoint, data from the checkpoint is read from operand queue 345 and is registered in the CAM registers of store forwarding array 351, as well as written into that slot's position in virtual commit buffer 350. The virtual commit pointer is then advanced to the next floating-point store. If that store operation is valid in operand queue 345, the data from that slot will be transferred from operand queue 345 into virtual commit buffer 350. Once all stores have been transferred to virtual commit buffer 350, FPU 230 asserts a signal, FPUStrStoreCommitOK, and also asserts exception status for the commit level. When the commit unit indicates that the

checkpoint should be committed, the data is transferred from virtual commit buffer 350 to write queue 355.

Sub A8 The virtual commit pointer is advanced as quickly as it can be through the slots in virtual commit buffer 350. This means that the virtual commit pointer does not wait for a store to complete for it to advance. Instead, as soon as a checkpoint has been issued to the load/store unit, the virtual commit pointer pulls all stores from operand queue 345 and forwards data from the store operation to any dependant read operation. The virtual commit pointer only stops after all stores for the three virtual commit checkpoints have been read.

When a store occurs, the store data is written into operand queue 345 at the address indexed by the store slot:checkpoint value and the CAMs in forwarding array 351 compare the store address with all "forward from" addresses so that all dependant reads will be updated as well. The CAM outputs are used as word lines for operand queue 345 and are also used to mark the dependant reads as needing re-execution. Store operations also write into virtual commit buffer 350 at the proper slot:checkpoint value, so that it is not necessary to back up the virtual commit pointer to the slot:checkpoint value where the store occurred.

If a store forwarding request is set up with a virtually committed source, the virtual commit pointer is backed up to the

offending slot. This allows the forwarded data to be read from operand queue 345 and to be written into the forwarded slot. It is not necessary to check destinations of store forwards since the "from" address must be less than the "to" address.

5 The virtual commit tag/exception unit 352 accumulates tags from the stores that have been virtually committed. The tags are only accumulated for the current checkpoint and are reset at the beginning of a checkpoint. The PRF commit logic uses the OR of all exception bits in these tags to determine if there is a pending exception that should be marked in the status register or should cause a commit fail.

FIGURE 5 depicts flow chart 500, which illustrates the operation of floating point unit (FPU) 230 containing operand queue 345 according to one embodiment of the present invention. During a load operation in which an operand data is retrieved from system memory 140 (i.e., from data cache 220), the incoming operand data are aligned, if necessary, in aligner units 401a and 401b. When the Valid High and Valid Low bits are both set for that operand queue entry, the operand is available for transfer to load conversion units 315a and 315b (process step 405).

20 During a store operation in which operand data are to be sent to system memory 140, the store data are stored and held in operand queue 345 and in virtual commit buffer 350 until the store data

instruction is finally committed. The operand queue address of the store operation instruction is stored in the CAM portion of forwarding array 351 in virtual commit buffer 350 (process step 410). The memory subsystem checks all subsequent load operations in FPU 230 against outstanding store operations in forwarding array 351 to determine if any of the subsequent load operations are dependent on the data operand associated with the outstanding store operation. If a subsequent load operation is dependent on the store data operation, then the address in operand queue 345 that holds the operand of the store operation is also written into forwarding array 351 at the address of the dependent load operation (process step 415). When the dependent load operation is subsequently executed, the address of the dependent load instruction is used as an index into forwarding array 351 to retrieve the address in operand queue 345 of the needed data (process step 420).

By way of example, suppose that FPU 230 holds sixteen operations. Operand queue 345 and virtual commit buffer 350 then each contain sixteen entries, one for each entry in pipelined FPU 230. The fifth instruction, I5, in FPU 230 is the operation "store address 2 from store conversion unit 317." The tenth instruction, I10, in FPU 230 is the operation "load address 2 to store conversion unit 315a." Instruction I10 (at address 10) is

dependent on the data from previous instruction I5 (at address 5). The memory subsystem detects the dependent load and causes FPU 230 to write the address value "5" into forwarding array 351 in two places: at address 5 and at address 10. The operand data associated with the store address operation is written into operand queue 345 at address 5 and at address 10. Now when instruction I10 is executed, the dependent load operation will be satisfied using the operand data from location 10 in operand queue 345. It is not necessary for the data to be retrieved from address 2 in system memory 140. Thus, the latency associated with a read-after-write (RAW) hazard is avoided, since it is no longer necessary to wait for the write operation to be completed to system memory 140.

Although the present invention has been described in detail, those skilled in the art should understand that they can make various changes, substitutions and alterations herein without departing from the spirit and scope of the invention in its broadest form.